Book Recommendation System — Collaborative & Content-Based Filtering Approaches

Milos Saric

GitHub LinkedIn saricmilos.com Real SKILLS Over DEGREES

November 11, 2025

Abstract

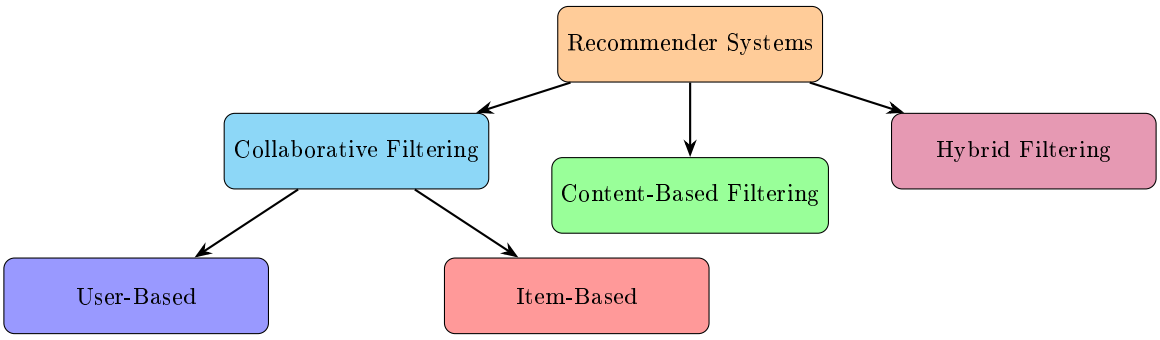
This report investigates the Kaggle Book Recommendation dataset to develop intelligent book recommendation systems using both collaborative filtering and content-based methods. A key application of machine learning is generating personalized recommendations for users, aiming to boost revenue, engagement, or other performance metrics. As a result, understanding how these recommendation algorithms work is an essential skill for any Machine Learning Engineer. Recommender systems have become a vital component of online platforms such as YouTube, Amazon, and Netflix, helping users discover content tailored to their preferences. By analyzing user behavior and item attributes, these systems predict user interests, enhancing user experience while driving engagement and revenue. This study highlights the implementation, advantages, and practical significance of recommendation algorithms in modern digital platforms.

---

1 Introduction and Problem Definition

Recommender systems aim to predict user preferences and suggest items they are most likely to enjoy. This project focuses on developing a **book recommendation system** using the Kaggle **Book-Crossing Dataset**, leveraging both **collaborative**, **content-based approaches** and **hybrid filtering approaches** to deliver personalized book suggestions. The goal is to design a system that accurately predicts and recommends books a user is likely to enjoy based on past interactions, ratings, and preferences. A clear understanding of this problem ensures that all subsequent analysis and modeling efforts align with the primary objective.

The main purpose of a recommender system is to boost product sales. Businesses use these systems to maximize their profits by suggesting products that customers are likely to be interested in. By highlighting relevant and appealing items, recommender systems help users discover products they might otherwise miss — ultimately driving higher sales and greater profits for the company.



1.1 Scope

The analysis focuses exclusively on the Kaggle dataset, which includes:

- **Users:** demographic and identification data.
- **Books:** metadata such as title, author, year, publisher, and cover images.
- **Ratings:** explicit ratings (1–10) and implicit feedback (0 for interactions without ratings).

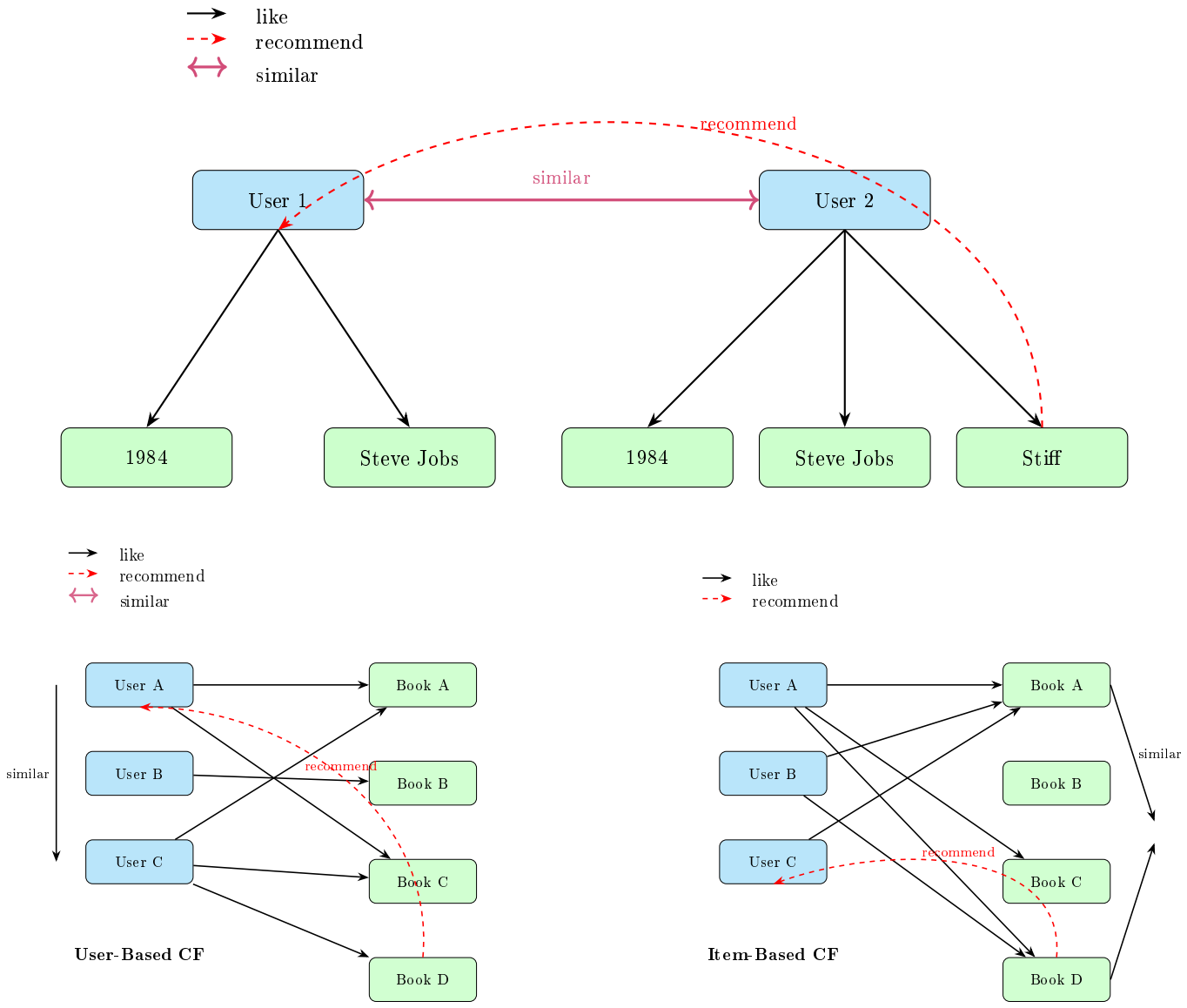
Predictions are restricted to this dataset without using external sources unless explicitly integrated in advanced phases.

1.2 Stakeholders

- **Readers / Users:** receive personalized book recommendations.
- **Publishers / Authors:** gain insights into reader interests for better targeting.
- **Data Scientists / ML Practitioners:** test and optimize recommendation algorithms.
- **Platform Developers / Businesses:** improve user engagement, retention, and revenue through effective recommendations.

2 Neighborhood-Based Collaborative Filtering

Collaborative filtering creates a model based on a user's past actions such as items they have purchased, selected, or rated, as well as the behavior of other users with similar preferences. This model is then used to recommend items or predict ratings for items that the user is likely to be interested in. Collaborative filtering is a recommendation method that predicts a users preferences based on the choices of other users with similar tastes. It relies on both user and item information, often organized in a user-item matrix.
This approach is widely used in industry across platforms like YouTube, Netflix and Amazon. For example, if users with similar past actions/tastes buy a product on Amazon, the system can suggest that same product to you. YouTube leverages this technique to recommend videos, Amazon uses it to suggest books and products, and Netflix relies heavily on collaborative filtering to personalize movie recommendations.



Collaborative filtering relies heavily on user data. As users continue to browse, purchase, and rate products, their preferences evolve over time. Because of this, the underlying recommendation model must be regularly updated and refined to stay accurate and relevant. In practice, this means continuously integrating new data and adjusting the system to reflect changing user behavior and market trends.
The two main sources of data for collaborative filtering are:

- **Explicit data:** Information that users actively provide, such as ratings, reviews, or responses to surveys and questionnaires.
- **Implicit data:** Information inferred from user behavior, including browsing history, clicks, time spent on content, likes, shares, and other engagement signals.

Both types of data are essential for understanding user preferences. Explicit ratings capture direct feedback, while implicit data reveals natural behavioral patterns that help improve recommendation accuracy over time.

2.1 User-Based Collaborative Filtering

User-based collaborative filtering recommends items to a user by identifying other users with similar preferences. If a similar user liked an item, it can be suggested. In our user based book recommendation system, we begin by constructing a matrix with users as rows and books (identified by ISBN) as columns let's call this the user-book matrix. Each entry in the matrix represents either a rating from 1 to 10, indicating that the user has read and rated the book, or a zero if the user has not interacted with that book. For example, if user J has rated book K, the corresponding cell contains that rating. If user J has not read or rated book K, the cell is simply zero.
User-based collaborative filtering recommends items to a user by identifying other users with similar preferences. If a similar user liked an item, it can be suggested.
Similarity between users can be measured using metrics such as **Cosine similarity**, **Pearson correlation**, or **Euclidean distance**. Read more about similarity at Section 4.

2.1.1 Algorithmic Workflow

1. Represent each user as a vector in item space:

   In user-based collaborative filtering, we start from a set of observed user-item ratings

   $$R = \{(u, i, r_{ui}) \mid u \in U,\, i \in I\} \tag{1}$$

   where U is the set of users, I is the set of items (books), and r_{ui} is the rating given by user u to item i.

   We represent this data as a **user-item rating matrix** $M \in \mathbb{R}^{|U| \times |I|}$:

   $$M_{ui} = \begin{cases} r_{ui}, & \text{if user } u \text{ rated item } i \\ 0, & \text{otherwise.} \end{cases} \tag{2}$$

   Each **row** of this matrix corresponds to a *user vector*:

   $$\mathbf{u}_k = [\, r_{k1}, r_{k2}, \ldots, r_{k|I|} \,] \in \mathbb{R}^{|I|} \tag{3}$$

   where r_{kj} is the rating of user k for item j, and I is the number of items. Thus, the moment we pivot our data from a triplet (u, i, r_{ui}) format into a matrix form, we have implicitly constructed a high-dimensional vector representation for every user.

2. Compute user-user similarity, e.g., using **cosine similarity**:

   $$\text{sim}(u, v) = \frac{\mathbf{u} \cdot \mathbf{v}}{\|\mathbf{u}\|\, \|\mathbf{v}\|} \tag{4}$$

   where $\mathbf{u} \cdot \mathbf{v}$ is the dot product, and $\|\mathbf{u}\|$ is the Euclidean norm:

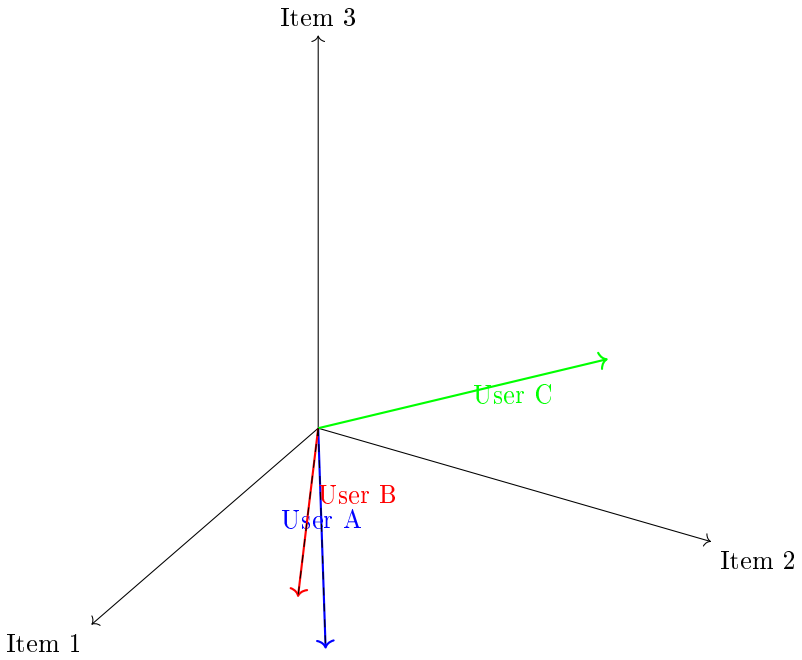
   $$\|\mathbf{u}\| = \sqrt{\sum_{j=1}^{M} r_{uj}^2} \tag{5}$$

   - Each user is a vector in M-dimensional space, where each axis represents an item.

- The length of the vector corresponds to the magnitude of their ratings.
- Cosine similarity measures the **angle between user vectors**, i.e., the similarity in rating patterns.
- Example:

$$\text{User A} = [5, 3, 0], \quad \text{User B} = [4, 2, 0]$$

Even though User B gives slightly lower ratings, the *pattern is similar*, resulting in a high cosine similarity.



Cosine similarity compares rating patterns, not absolute rating levels. Length of the vector = total "strength" of user ratings, but cosine similarity is normalized so that only direction matters.

3. Predict ratings for a user u on item i using top-K similar users $N(u)$:

There are two related but different formulas commonly used for user-based collaborative filtering (CF).

The correct and most widely used formula for predicting a user's rating is:

$$\hat{r}_{ui} = \bar{r}_u + \frac{\sum_{v \in N(u)} \text{sim}(u, v) \cdot (r_{vi} - \bar{r}_v)}{\sum_{v \in N(u)} |\text{sim}(u, v)|} \tag{6}$$

- $\hat{r}_{ui}$ is the *predicted rating* of user u for item i.
- $\bar{r}_u$ represents the *average rating* of user u (the user's rating bias).
- $N(u)$ denotes the set of *nearest neighbors* (similar users to u) who have rated item i.
- r_{vi} is the *rating given by neighbor* v to item i.
- $\bar{r}_v$ is the *average rating* of user v.
- $\text{sim}(u, v)$ measures the *similarity* between users u and v, often computed using cosine similarity, Pearson correlation, or other metrics.

A simpler, less accurate version of the formula is sometimes used:

$$\hat{r}_{ui} = \frac{\sum_{v \in N(u)} s(u, v) \cdot r_{vi}}{\sum_{v \in N(u)} |s(u, v)|} \tag{7}$$

- This version ignores user rating biases ($\bar{r}_u$ and $\bar{r}_v$).
- It assumes all users rate on the same scale, which is often not true in practice.

- Although simpler, it generally performs worse in predicting ratings accurately.

In summary, the mean-centered formula is preferred for more accurate and bias-adjusted predictions, while the simpler one can be used as a baseline or when user averages are unavailable.

4. Evaluation of Collaborative Filtering Predictions:

   Evaluating the performance of a collaborative filtering (CF) model involves measuring how accurately the system predicts user ratings and how effective its recommendations are. Two major evaluation perspectives are commonly used: prediction accuracy and recommendation quality.

5. **Evaluation Procedure**

   A standard workflow:

   (a) Split the dataset into training and test sets (e.g., 80/20 split).

   (b) Build the CF model using the training set.

   (c) Predict ratings $\hat{r}_{ui}$ for test set user–item pairs.

   (d) Compare predicted and actual ratings using MAE, RMSE, etc.

   (e) Optionally, evaluate top-N recommendations using Precision@N and Recall@N.

2.1.2 Summary

- Each user is a vector in an $|I|$-dimensional space (items are dimensions).
- Ratings are vector components (coordinate values).
- Similarities are computed between users' vectors.
- Predictions are weighted averages of neighbors' ratings.

2.2 Item-Based Collaborative Filtering

Item-based collaborative filtering recommends items to a user by analyzing similarities between items rather than users, based on user interactions. The intuition is: if a user liked a particular item, they are likely to enjoy other items that received similar ratings from other users. For example, in a book recommendation system, if two books are rated similarly by many users, they are considered similar, and a user who liked one might like the other.

Matrix Representation The cosine similarity function itself does not know whether we are computing user-based or item-based similarity. The distinction depends solely on the orientation of the rating matrix M. We have two perspectives:

1. **User-Based Filtering:** Each user is represented as a vector in the *item space*. This yields a user–user similarity matrix of shape $|U| \times |U|$.
2. **Item-Based Filtering:** Each item is represented as a vector in the *user space*. This yields an item–item similarity matrix of shape $|I| \times |I|$.

In practice, this difference is implemented by transposing the matrix before applying cosine similarity:

$$\text{user-based: cosine\_similarity}(M), \quad \text{item-based: cosine\_similarity}(M^T).$$

Each item becomes a vector in *user space*, with dimensions corresponding to users:

$$\mathbf{i}_j = [r_{1j}, r_{2j}, r_{3j}, \ldots, r_{Nj}] \in \mathbb{R}^{|U|} \tag{8}$$

where r_{uj} is the rating given by user u to item j, and $N = |U|$ is the number of users. The transposition means that the same rating matrix M can be used, but similarity computations are performed along the columns instead of rows.

Item Similarity Similar to user-based CF, we compute the similarity between item vectors using metrics like cosine similarity:

$$\text{sim}(i,j) = \frac{\mathbf{i}_i \cdot \mathbf{i}_j}{\|\mathbf{i}_i\| \, \|\mathbf{i}_j\|} \tag{9}$$

This produces an $|I| \times |I|$ item-item similarity matrix. The similarity values are then used to predict a user's rating for a target item based on the ratings of similar items they have already rated.

Prediction Formula The predicted rating of user u for item i is computed as:

$$\hat{r}_{ui} = \frac{\sum_{j \in N(i)} s(i,j) \cdot r_{uj}}{\sum_{j \in N(i)} |s(i,j)|} \tag{10}$$

where:

- $N(i)$ is the set of top-K items similar to i that user u has rated.
- $s(i,j)$ is the similarity between items i and j.
- r_{uj} is the rating of user u for item j.

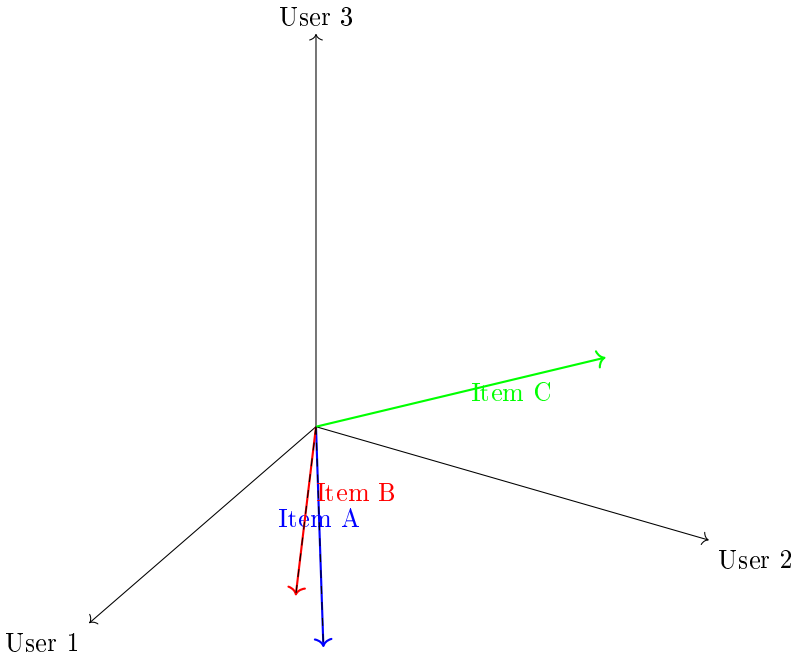
This formula is analogous to the user-based CF prediction, but here the weights come from item-item similarity rather than user-user similarity.

Algorithmic Workflow

1. Represent each item as a vector in user space ($|U|$-dimensional).
2. Compute the item-item similarity matrix using cosine similarity.
3. For each user and target item, identify the set of similar items the user has rated ($N(i)$).
4. Predict the rating using a weighted sum of the user's ratings on similar items.
5. Optionally, generate top-N recommendations based on predicted ratings.

Intuition and Advantages

- Each item is a vector in user space, analogous to users being vectors in item space for user-based CF.
- Cosine similarity measures which items are rated similarly across users.
- IBCF often performs better in sparse datasets, as items tend to have more ratings than individual users, making similarity computations more robust.
- Reduces computational cost when the number of items is smaller than the number of users.



3 Content Based Filtering

Content-Based Filtering recommends items that are *similar to those a user has previously liked.* It relies on the features or attributes of the items, such as movie genre, product category, author, or keywords. The system first builds a user profile based on the characteristics of items the user interacted with, then suggests other items with similar properties. It generates recommendations by analyzing the features of items and matching them to a user's preferences. Essentially, the system compares a **user profile** with an **item profile** to predict which items the user is likely to engage with. Content-Based Recommender Systems often build a personalized model for each user. The process begins by collecting the features of items the user has interacted with which forms the user profile. These items serve as a training set for a user-specific classifier or regression model.
In the model, the item features act as independent variables, while the user's past behavior-such as ratings, likes, or purchases serve as the dependent variable. Once trained, the model predicts the user's likely response to new items, allowing the system to recommend items that align with the user's preferences. The **item profile** captures the characteristics of an item, including structured features or descriptive metadata. For example, a streaming service may represent movies using attributes such as genre, director, release year, or cast.
The **user profile** reflects the individual's preferences and past behavior. It is typically built from items the user has previously interacted with, including ratings, likes, dislikes, searches, or other engagement data. In short, this approach focuses on *what the user likes*, rather than what other users like.

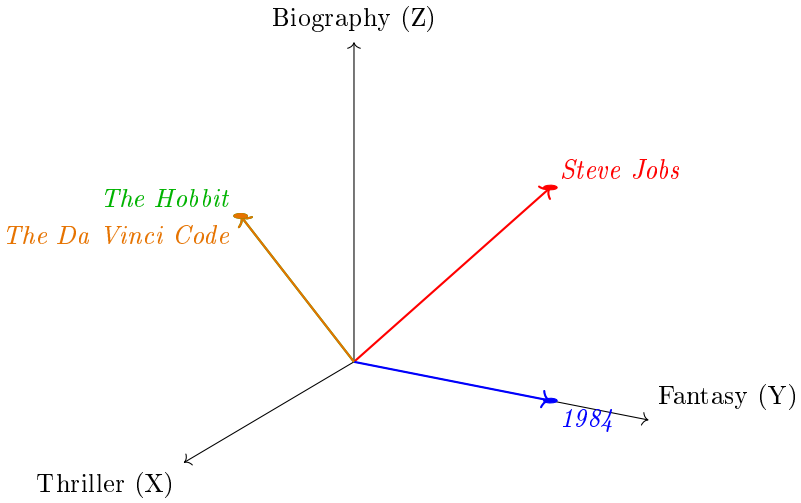
3.1 Item Representations in Content-Based Filtering

In content-based filtering, items and users are often represented as vectors in a multi-dimensional feature space. Each item's feature such as genre, author, or other metadata define its coordinates. Boolean values (1 for presence, 0 for absence) are commonly used to indicate whether an item possesses a particular feature.
Example: A few novels represented by three genres (Biography, Fantasy, Thriller):

Table 1: Boolean Feature Representation of Novels

Novel	**Biography**	**Fantasy**	**Thriller**
Steve Jobs	0	1	1
1984	0	1	0
The Hobbit	1	0	1
The Da Vinci Code	1	0	1

In this 3D vector space, each dimension corresponds to a feature (Adventure, Bildungsroman, Children). A novel's coordinates are determined by its Boolean values. For example:



The closer two novels are in this space, the more similar they are according to the selected features. For instance, *The Hobbit* and *The Da Vinci Code* overlap completely, while *Steve Jobs* is closer to *1984* than to the thriller novels. Since The Hobbit and The Da Vinci Code are very similar in the feature space, a user who has previously purchased The Hobbit is likely to be recommended The Da Vinci Code, as it closely matches their interests.

Visualization: We can plot these items in a 3D Cartesian coordinate system, with each axis representing one genre. Novel vectors are points in this space, and proximity reflects similarity. Adding more features (e.g., Fantasy, Gothic) increases the dimensionality, adjusting item positions accordingly.

In content-based filtering, each user is represented by a vector that summarizes their preferences. This user vector, denoted $\mathbf{u}_x$, is typically computed as a weighted average of the vectors of the books the user has liked or rated highly:

$$\mathbf{u}_x = \frac{1}{N_x} \sum_{i \in I_x} w_{xi} \cdot \mathbf{v}_i \tag{11}$$

where I_x is the set of books user x has interacted with, w_{xi} is the weight associated with book i for this user (such as a normalized rating or binary like/dislike), $\mathbf{v}_i$ is the feature vector of book i, and N_x is a normalization factor.

Once the user vector is constructed, the next step is to compute the similarity between this user vector and the vector of every book in the catalog. The most common similarity measure is cosine similarity, defined as

$$\text{sim}(\mathbf{u}_x, \mathbf{v}_i) = \frac{\mathbf{u}_x \cdot \mathbf{v}_i}{\|\mathbf{u}_x\| \, \|\mathbf{v}_i\|} \tag{12}$$

High similarity indicates that the book aligns closely with the user's preferences, making it a strong candidate for recommendation. Low similarity suggests the book is less likely to match the user's interests. After computing the similarity for all books, they are sorted by score, and the top-K books are recommended to the user, excluding those they have already rated. This process allows the system to suggest books that are most likely to match the user's tastes based on the features encoded in the vectors.

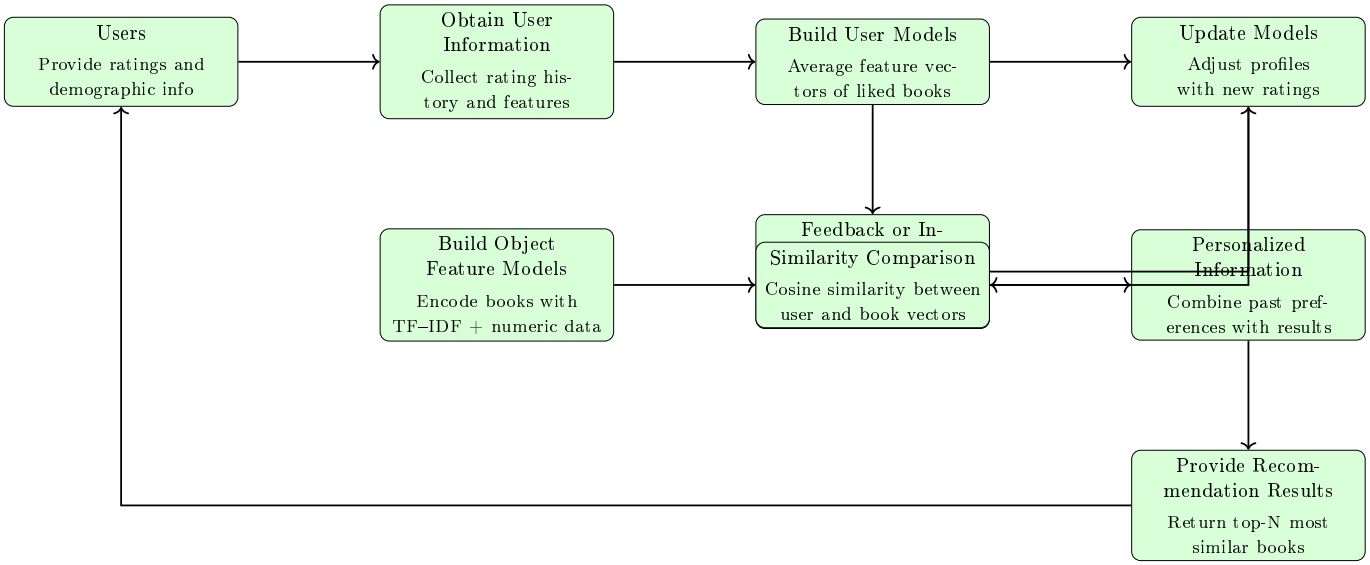


Figure 1: Flowchart of the Content-Based Filtering process with steps implemented in the book recommender system.

3.2 Item-Based Collaborative Filtering vs. Content-Based Filtering

Although both methods aim to recommend items similar to those a user already liked, they differ in what information they rely on and how they define similarity.

Item-Based Collaborative Filtering (CF) focuses on *user behavior patterns*. Items are considered similar if many users have interacted with both of them in similar ways. For example, if several users who bought *Book A* also bought *Book B*, the system concludes that these two books are related, even if their topics are entirely different. This method depends on user ratings or implicit feedback (such as clicks or purchases) to measure similarity between items. In essence, the approach follows the idea: *"Users who liked this item also liked that item."*

Content-Based Filtering (CBF), on the other hand, focuses on *the characteristics of the items themselves*. It recommends new items similar to those the same user has already liked, based on item features such as genre, keywords, author, or description. For instance, if a user enjoys a mystery novel with a strong female lead, the system suggests other mystery novels with similar themes or features. The guiding principle is: *"You liked this because of its content, so you might like similar content."*

3.3 Advantages and Disadvantages of Collaborative and Content-Based Filtering

Collaborative Filtering: Advantages: Collaborative filtering can uncover new items that a user may not have considered before by leveraging the preferences of similar users. This allows for more diverse recommendations that go beyond the user's past interactions.

Disadvantages: Collaborative filtering suffers from the *cold-start problem*. New users have no interaction history, making it difficult to find similar users, and new items lack sufficient ratings to be recommended effectively. Additionally, calculating similarities between users can be computationally expensive for large datasets.

Content-Based Filtering: Advantages: Content-based filtering handles new items efficiently, as recommendations rely on item features rather than prior user interactions. It also provides greater transparency, explaining why a particular item is recommended. For example, a movie may be suggested because it shares the same genre or actors as previously liked movies, allowing users to make informed decisions.

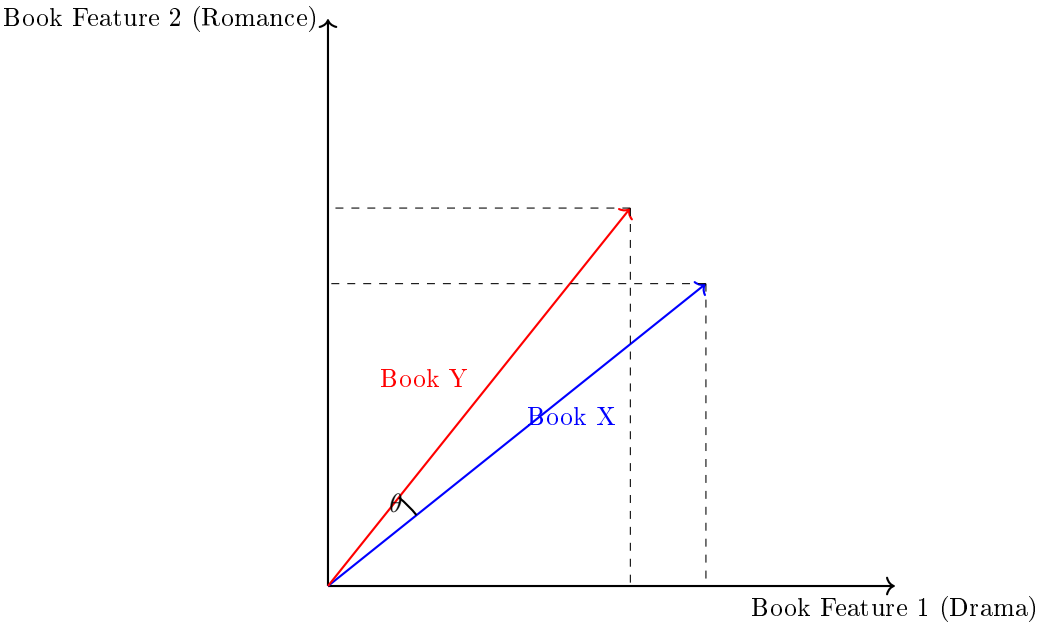
Disadvantages: Content-based filtering is limited by the features available to describe items. If a user's preference is based on characteristics not captured in the item profile (e.g., specific plot elements or production details), the system may fail to provide relevant recommendations. Moreover, it tends to overspecialize, suggesting items very similar to those already liked, which reduces diversity and limits discovery of new or unexpected items.

4 Similarity/Distance Measures

In content-based filtering, items are compared using their feature representations, and those that are closer in feature space are considered more similar. In collaborative filtering, similarity is computed between users or items based on their interaction or rating patterns, with closer vectors indicating stronger similarity. Both approaches use metrics like cosine similarity or correlation to quantify closeness.

4.1 Cosine Similarity

One of the most common methods, especially for high-dimensional feature spaces, is **Cosine Similarity**. Cosine similarity measures cosine of the angle between two vectors, giving a value between -1 and 1. The closer the value is to 1, the more similar the items are considered.



Here, θ is the angle between vectors $\mathbf{x}$ and $\mathbf{y}$. Cosine similarity measures how aligned these vectors are: the smaller the angle, the closer the similarity is to 1.
Intuition: Cosine similarity focuses on the direction of vectors rather than their magnitude. Two items with similar feature patterns are considered similar, even if one has generally higher values than the other.
The formula for cosine similarity between two item vectors $\mathbf{x}$ and $\mathbf{y}$ is:

$$\text{sim}(\mathbf{x}, \mathbf{y}) = \frac{\mathbf{x} \cdot \mathbf{y}}{\|\mathbf{x}\| \, \|\mathbf{y}\|} = \frac{\sum_{i=1}^{n} x_i y_i}{\sqrt{\sum_{i=1}^{n} x_i^2} \, \sqrt{\sum_{i=1}^{n} y_i^2}} \tag{13}$$

Here, $\mathbf{x} \cdot \mathbf{y}$ represents the dot product of the two vectors, where $\mathbf{x}$ and $\mathbf{y}$ are vectors, for example, item feature vectors in a recommendation system and $\|\mathbf{x}\|$ and $\|\mathbf{y}\|$ are the magnitudes (lengths) of the vectors. This metric allows the system to quantify similarity between items and recommend those that are most closely aligned with the user's previous preferences.
In content-based filtering, each item is represented as a vector in a multi-dimensional feature space. For example, a book could be represented as

$$\mathbf{x} = [\text{drama score}, \text{romance score}, \text{adventure score}, \ldots].$$

- The **dot product** of two vectors $\mathbf{x}$ and $\mathbf{y}$ quantifies how much the vectors point in the same direction:

$$\mathbf{x} \cdot \mathbf{y} = x_1 y_1 + x_2 y_2 + \cdots + x_n y_n = \sum_{i=1}^{n} x_i y_i \tag{14}$$

  Each term represents the contribution of a feature to the overall similarity.

- The **norm** or length of a vector is:

$$\|\mathbf{x}\| = \sqrt{x_1^2 + x_2^2 + \cdots + x_n^2} \tag{15}$$

  Normalizing by the vector lengths ensures that cosine similarity measures direction rather than magnitude.

Cosine similarity is then defined as:

$$\text{sim}(\mathbf{x}, \mathbf{y}) = \frac{\mathbf{x} \cdot \mathbf{y}}{\|\mathbf{x}\| \, \|\mathbf{y}\|} = \cos\theta \tag{16}$$

where θ is the angle between the vectors. The similarity ranges from -1 (opposite) to 1 (identical), with 0 indicating orthogonal (no similarity).

4.1.1 Euclidean Distance

Euclidean distance is a way to measure how far apart two items are in a multi-dimensional feature space. It is the length of the straight line connecting the two points representing the items. In recommendation systems, a smaller Euclidean distance between two items indicates that they are more similar in terms of their features.
Suppose we represent two books as feature vectors:

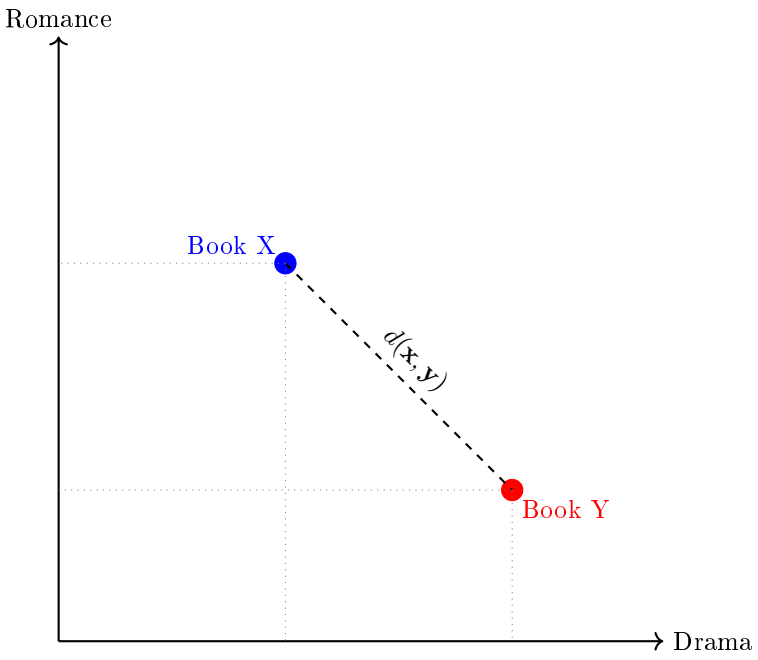
$$\mathbf{x} = (\text{drama score}, \text{romance score}, \dots), \quad \mathbf{y} = (\text{drama score}, \text{romance score}, \dots)$$

For n features, the Euclidean distance between $\mathbf{x}$ and $\mathbf{y}$ is:

$$d(\mathbf{x}, \mathbf{y}) = \sqrt{(\text{drama}_x - \text{drama}_y)^2 + (\text{romance}_x - \text{romance}_y)^2 + \dots} = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2} \tag{17}$$

- Each term $(x_i - y_i)^2$ measures the squared difference in a single feature (e.g., drama, romance) between the two books.
- Summing over all features gives a combined measure of difference across all dimensions.
- Taking the square root converts this sum into the actual straight-line distance.

Intuition: - Two books with similar ratings across all features will have a **small Euclidean distance**. - Books with very different ratings in one or more features will have a **larger distance**.



This diagram shows:

- Each book as a point in a 2D feature space (Drama vs Romance).
- The dashed line representing the Euclidean distance between the books.
- Dotted lines showing the projections to each feature axis for clarity.

4.2 Pearson Correlation

Pearson correlation measures the **linear relationship** between two items across multiple features. Unlike cosine similarity or Euclidean distance, Pearson correlation **removes the effect of magnitude** by centering each vector on its mean. This makes it useful when we care about **rating patterns** rather than absolute values.

DEFINITION Given two books represented as feature vectors:

$$\mathbf{x} = [\text{drama}_x, \text{romance}_x, \dots], \quad \mathbf{y} = [\text{drama}_y, \text{romance}_y, \dots]$$

Let $\bar{x}$ and $\bar{y}$ be the mean values of the features of each book:

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i, \quad \bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_i$$

The Pearson correlation is:

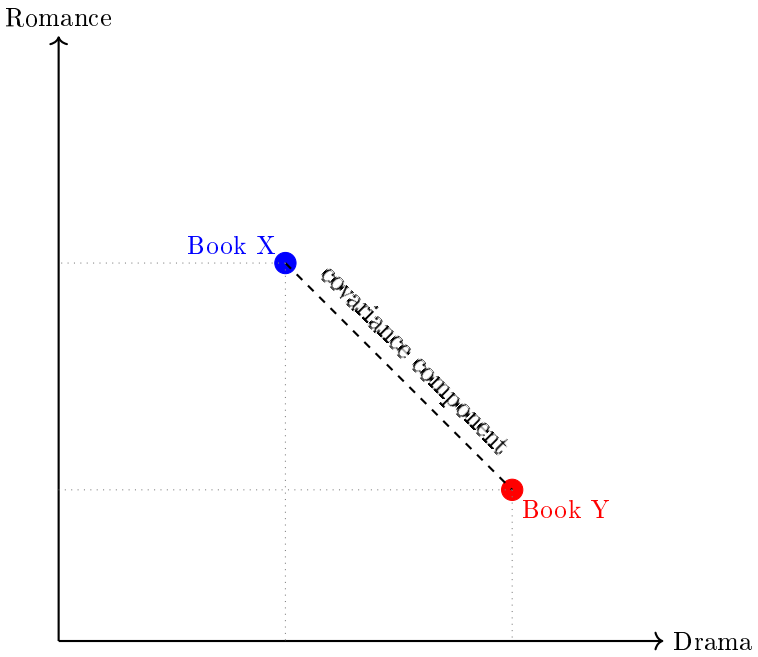
$$\text{corr}(\mathbf{x}, \mathbf{y}) = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2}\,\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}} \tag{18}$$

- Each feature is **mean-centered**: we subtract the book's average feature value.
- The numerator computes the **covariance** between the two books' features.
- The denominator normalizes by the magnitude of the mean-centered vectors, giving a value between -1 and 1.

INTERPRETATION

- $+1 \rightarrow$ features vary together perfectly (strong positive correlation).
- $0 \rightarrow$ no linear relationship between features.
- $-1 \rightarrow$ features vary in opposite directions (strong negative correlation).

Example in 2D feature space: Drama vs Romance



Notes:

- The vectors are **mean-centered** by subtracting each book's average feature value.
- The dashed line visually represents the **co-movement of features** contributing to Pearson correlation.
- This method measures **pattern similarity**, so two books with similar relative features but different absolute levels can still have a high correlation.

5 Evaluation Metrics

To assess the performance of the content-based recommender system, we employ two categories of evaluation metrics: (1) prediction accuracy metrics, which measure how accurately the system predicts user ratings, and (2) top-N recommendation metrics, which evaluate the quality of the ranked lists of recommended books.

5.1 Prediction Accuracy Metrics

These metrics evaluate how close the predicted ratings $\hat{r}_{ui}$ are to the actual user ratings r_{ui}.

- **Mean Absolute Error (MAE)**

$$MAE = \frac{1}{|T|} \sum_{(u,i) \in T} |r_{ui} - \hat{r}_{ui}| \tag{19}$$

  where T denotes the test set of user–item pairs. Lower MAE indicates higher predictive accuracy.

- **Root Mean Squared Error (RMSE)**

$$RMSE = \sqrt{\frac{1}{|T|} \sum_{(u,i) \in T} (r_{ui} - \hat{r}_{ui})^2} \tag{20}$$

  RMSE penalizes larger errors more heavily than MAE, and thus reflects how severely the system deviates from the actual ratings.

- **Mean Squared Error (MSE)**

$$MSE = \frac{1}{|T|} \sum_{(u,i) \in T} (r_{ui} - \hat{r}_{ui})^2 \tag{21}$$

  MSE serves as the base metric for RMSE, providing a measure of the average squared deviation between predicted and true ratings.

- **Normalized Mean Absolute Error (NMAE)**

$$NMAE = \frac{MAE}{r_{\max} - r_{\min}} \tag{22}$$

  NMAE normalizes the MAE by the range of the rating scale, enabling comparability across datasets with different rating intervals.

5.2 Top-N Recommendation Metrics

When the system produces a ranked list of recommended items, ranking-based metrics are preferred over pure rating prediction metrics.

- **Precision@N**

$$Precision@N = \frac{\text{Number of relevant items in top } N}{N} \tag{23}$$

  Precision@N measures the proportion of relevant items among the top-N recommendations.

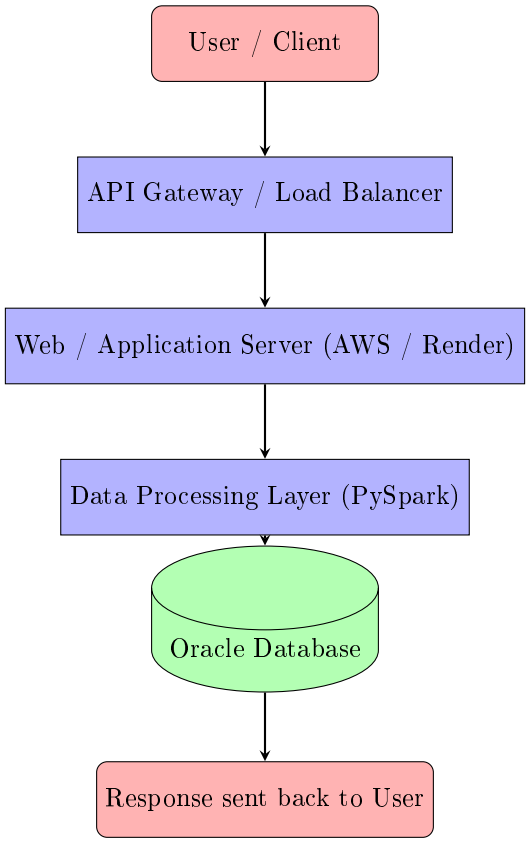
- **Recall@N**

$$Recall@N = \frac{\text{Number of relevant items in top } N}{\text{Total number of relevant items}} \tag{24}$$

  Recall@N evaluates how well the system retrieves all relevant items for a user, indicating the completeness of recommendations.

6 Deployment of Book Recommendation System

Having an app on our local system simply isn't enough, it needs to be accessible online. Deployment is the process of making your application available for users so they can access and interact with it. It usually means moving your code from a development environment to a live production environment, where the application is fully operational.



6.1 Application Programming Interface (API)

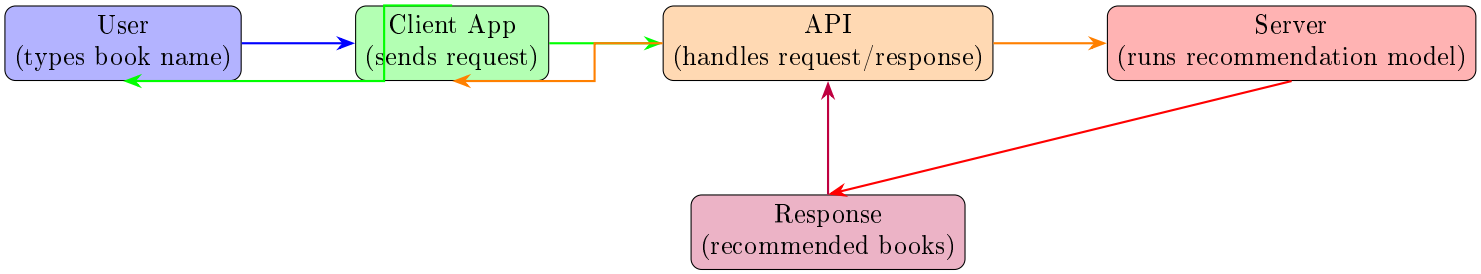
An API, short for Application Programming Interface, is basically a bridge that lets different software programs talk to each other. It defines a set of rules and protocols that allow applications to exchange data, features, or functionality. Instead of building every feature from scratch, developers can use APIs to connect their apps with existing services or data sources, saving time and effort. APIs also give application owners a controlled and secure way to share certain parts of their app's data or capabilities—either with internal teams or with outside users. They're designed to expose only the specific information or actions that are needed for a given task, not the entire system. By doing this, APIs help protect sensitive information and maintain overall system security while still enabling smooth, efficient communication between different software systems.

It helps to think of API communication as a conversation between a client and a server. In a book recommender system, the user's app—the client—sends a request, and the server responds with recommendations. The API is the bridge that makes this connection possible.

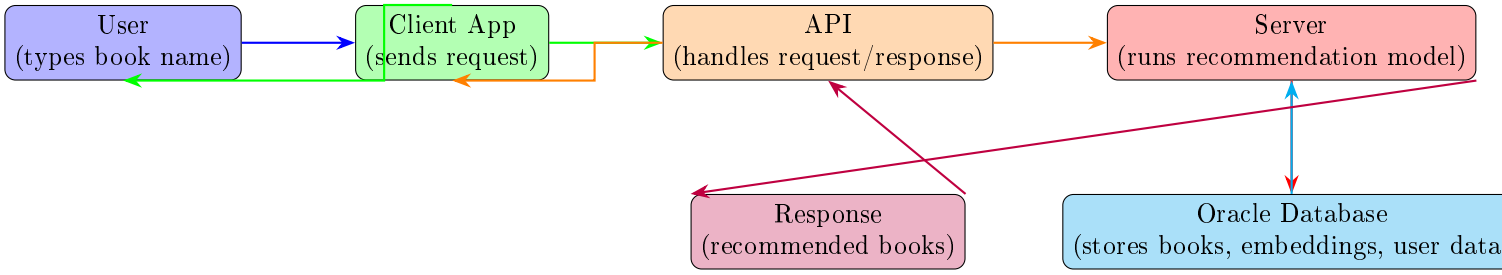
Here's how it works in practice: imagine a user types in the name of a book they like. When they hit "Submit," the app sends this information as a request through the API. The request travels to the server, which runs a trained recommendation model to find books similar to the one the user entered.

The server then sends back a response containing a list of recommended books. The API takes this data and delivers it to the app, so the user sees their personalized recommendations almost instantly.

From the user's perspective, it feels seamless—they just type a book name and get suggestions. Behind the scenes, though, the API is carefully handling the request and response, sharing only the information needed, keeping the system secure, and making sure the app and server communicate smoothly.



1. **User Input:** The user types a book title (and optionally a user ID) in the application interface.
2. **Client App:** The client app collects the input and formats it into an API request (e.g., JSON) that is sent to the API.
3. **API:** The API acts as a gateway. It receives the request, validates it, and forwards it to the server. It also ensures that only necessary information is exchanged, keeping internal details secure.
4. **Server:** The server runs the recommendation logic:
   - Looks up the book embedding from the Oracle database or computes it if it's free text.
   - Performs a nearest-neighbor search in the embedding space to find similar books.
   - Optionally combines item similarity with user profile data for personalization.
5. **Oracle Database:** Stores all persistent data, including books, embeddings, and user interaction history. The server queries it using SQL, retrieves the necessary data, and processes it in memory.
6. **Response:** After computing the Top-N recommended books, the server sends the results back to the API, which forwards them to the client app. The user sees the recommendations seamlessly in the interface.



Notes on the Flow

- **User → Client → API → Server → Database:** This is the request path.
- **Database → Server → API → Client → User:** This is the response path.
- The API acts as a secure bridge and ensures that internal server and database logic is not exposed to the client or user.
- The Oracle database stores all persistent data, and the server retrieves only what is needed for computing recommendations.
- The server performs the actual recommendation computation, including vector lookups, nearest-neighbor searches, and optional personalization.